Computer Science Department

TECHNICAL REPORT

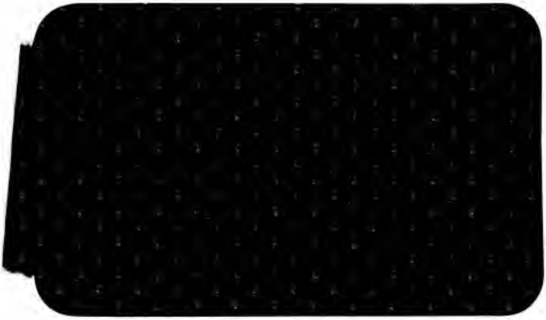
A Preliminary Evaluation of Trace Scheduling for Global Microcode Compaction

by

Ralph Grishman and Su Bogong *

May 1982

Report No. 043



A Preliminary Evaluation of Trace
Scheduling for Global Microcode Compaction

by

Ralph Grishman and Su Bogong *

May 1982

Report No. 043

* on leave from Tsing Hua University, Beijing

A Preliminary Evaluation of Trace Scheduling for Global Microcode Compaction

Ralph Grishman and Su Bogong *
Dept. of Computer Science
Courant Institute of Mathematical Sciences
New York University

ABSTRACT

Fisher has recently described a new procedure for global microcode compaction which he calls "trace scheduling." We have implemented this procedure and tested it on several microcode sequences. We report in this correspondence on the relative effectiveness of local compaction, manual compaction, and trace scheduling on these sequences.

* on leave from Tsing Hua University, Beijing

Fisher [1] recently described a new, relatively complex procedure for global microcode compaction which he calls "trace scheduling". To evaluate the efficacy of this algorithm, we have implemented it and tested it on several microcode sequences.

THE PROBLEM

To take advantage of the parallelism inherent in a horizontally microprogrammed machine, it is necessary to convert sequential microcode into equivalent parallel microcode. This task is called microprogram compaction or optimization. Because the task often involes interweaving logically separate code sequences, it is a tedious and error-prone operation. As a result, there has been considerable research over the past few years on automating this process.

Most of this research has focussed on local compaction -- the compaction of individual basic blocks [2]. However, because basic blocks in microcode are typically rather short, efficiencies approaching those of hand-written microcode can be achieved only by a procedure which is willing to move micro-operations from one basic block to another -- global compaction.

TRACE SCHEDULING

Earlier procedures for global microcode compaction have been based on a menu of rules for moving micro-operations from one basic block to another [3]. Such procedures can involve extensive tree searches (trying alternative sequences of microcode motions) and hence be very costly.

Fisher has proposed trace scheduling as an alternative approach. In essence, trace scheduling begins by identifying the most frequently traversed path through a section of microcode. A local compaction procedure is applied to this path, scheduling branch micro-operations just like other micro-operations (within data precedence constraints). Because arithmetic micro-operations may be moved relative to branches, a bookkeeping phase is required after compaction to "fix up" the microcode so that it is equivalent to the original (this primarily involves inserting duplicates of moved micro-operations into paths which enter or leave the main path). The procedure is then repeated on the main path through the code which remains uncompacted. If the code contains loops, the procedure will be first applied recursively to compact each loop.

IMPLEMENTATION

We have implemented most of the procedures described in Fisher [1], including the Schedule and Bookkeep routines, and all of the subroutines they invoke; the extensions needed for scheduling code with loops* and for handling equal edges in the data precedence graph. We have not automated the Picktrace algorithm, which selects the next path to schedule; instead, we have specified manually the order in which paths are to be considered. We have also not implemented the rules which Fisher grouped under "enhancements", such as space saving, task lifting, and his rules R2 and R4.

The implementation has been done in the "very-high-level" language SETL [4] running on a VAX-11/780. The use of SETL has speeded implementation and resulted in a rather short, although very slow, program (the program is approximately 3000 lines long and takes about 7 minutes to schedule a microcode segment of 49 micro-operations).

EXPERIMENTS

We at N.Y.U. had previously designed and built a horizontally microprogrammed emulator for the Control Data 6600 central processor [5]; three of these machines, dubbed PUMAs, are currently in operation. The microcoding of the original PUMA provided the impetus for Fisher's research, and Fisher used two PUMA microcode sequences as examples in his dissertation, where he worked through his algorithms by hand. We decided to check one of Fisher's examples on our implementation, and then to try two additional, somewhat longer, PUMA code segments. As one might expect, the most complex code sequences -- and hence the most challanging for compaction -- are those for floating-point arithmetic. Fisher chose as his examples the microcode for the normalize instruction and a portion of the floating multiply. We added to these the sequences for floating addition and division. The addition sequence has 42 micro-operations, including 11 conditional jumps; the division sequence 49 micro-operations and 15 conditional jumps.

As our benchmarks for evaluating trace scheduling we used the "production" PUMA microcode. This was very carefully hand-coded and reviewed by several readers, and is therefore probably optimally compacted or nearly so. To provide the input to the scheduler we rewrote the selected sequences as sequential PUMA microcode. This was not just a process of serializing parallel code. We went back to

---

* Except that our implementation, after placing a loop representative into a cycle, does not attempt to place any additional micro-operations into that cycle.

flowcharts for the arithmetic operations and tried to write the clearest possible sequential code, without regard for subsequent compaction. The flow graphs of the resulting sequential microcode are substantially different from those for the production (compacted) code. Nonetheless, we recognize that the fact that we wrote the sequential code with an awareness of the parallel code does introduce a possible bias in favor of the compaction algorithm.

One machine-specific optimization was performed by hand before submitting the sequential code to the compaction procedure: if a conditional branch tests the output of a register which was set by the previous micro-operation, these two operations can be combined into a single microinstruction with a conditional branch testing the <u>input</u> to the register (for many, but not all, conditional branches in the PUMA testing the output of a register there are corresponding operations testing the register's input). Since this is a local optimization, it should not be very difficult to automate.

As a consequence of moving conditional branches relative to other operations, Fisher's compaction procedure may generate many copies of a micro-operation in the initial code. Often there will be alternative schedules which are as fast (or nearly as fast) but require less duplication of micro-operations (and hence less space in the microprogram store). Fisher suggests some automatic "space saving" techniques for finding such schedules; we have not implemented these. However, to prevent the motion of conditional branches (and hence reduce code duplication) in some cases where we believed that the motion would not improve the schedule, we have manually added some edges to the data precedence graph generated by Fisher's algorithm.

We have also included a rule which avoids a substantial amount of the code duplication which Fisher would perform and later undo with his rules R2 and R4. We frequently have a situation in the microcode where a path forks (at a conditional jump) into two basic blocks (call them A and B) which subsequently rejoin. When a trace including block A is scheduled, a micro-operation <u>m</u> may be moved from above the fork to below the rejoin, or vice versa. In general, this will entail (during the "bookkeeping phase") adding micro-operations to block B and moving the rejoin point. However, if (using Fisher's terminology), the union of <u>readreg</u>, <u>writereg</u> and <u>condreadreg</u> of micro-operation <u>m</u> does not intersect the <u>readreg</u> or <u>writereg</u> of any micro-operation in B, <u>m</u> may be moved without any associated bookkeeping.

Finally, in order to obtain some measure of the relative advantage of global over local compaction, we manually performed a local compaction of these code sequences (i.e., code was moved only within a basic block, not between blocks).

RESULTS

Table 1 summarizes the results of the compaction procedure. For purposes of analysis, we have divided the floating addition into three parts (initialization, coefficient shift, and add) and similarly the floating divide (initialization, divide loop, and normalization and rounding). We successfully reproduced Fisher's results for the multiply initialization sequence, and repeat those results here.* The timing data represent weighted averages based on estimates of the relative frequency of the various paths.

The actual code sequences -- sequential, hand compacted, machine compacted (trace scheduled), and locally compacted -- are shown in Appendix A, and more detailed timing data is given in Appendix B.

DISCUSSION

The results of trace scheduling compared quite favorably with the hand-compacted code. The shift loop in the floating add and one path of the divide loop were one cycle longer in the trace scheduled versions; the initialization of the floating add was two cycles longer. All other code segments were compacted as well as the hand-coded versions (in the floating divide, the split of microoperations between pre- and post-loop is different in the two versions, but the total time was slightly shorter (by perhaps 0.1 cycle) for the machine compacted code). The locally compacted code was in most cases substantially slower than either the hand compacted or trace scheduled code, thus confirming the need for some global scheduling strategy.

The reason for the differences between the timings of the hand compacted and trace scheduled code in the shift and divide loops is readily explained. In each case in the sequential code the conditional jump is at the beginning of the loop, with an unconditional jump at the end back to the beginning. This structure is preserved in the machine-compacted code. In the hand-coded version, the conditional branch is replicated at the end of the loop, thus avoiding the unconditional branch. We could incorporate such a specialized optimization into our machine compaction procedure. More ambitiously, we could develop a procedure to unroll (replicate) a loop, schedule the

---

* Because our procedure does not incorporate space saving, our multiply sequence involved more code duplication than Fisher's result, but the timing of the main path in the two versions was identical.

unrolled loop (possibly moving an operation from one iteration to another), and then reroll the loop (identifying repeating code segments) [3]. Such a procedure should be able to perform the optimization just cited.

The difference in the initialization segment of the floating add is more complex. At one point this code forks, with one path interchanging two registers, the other not doing so. The hand-compacted code <u>inserts</u> into the latter path two successive interchanges (an identity) and then moves the interchange now shared by both paths to before the fork. We do not see how this transformation could be readily incorporated into an automatic compacter.

We are encouraged that, except for this last instance, the trace scheduler performs or can be readily extended to perform as well as a skilled microprogrammer. We look forward to more extensive tests of trace scheduling and in particular to evaluations of the space-saving procedures suggested by Fisher.

ACKNOWLEDGEMENTS

Computer facilities for these experiments were supported by the Department of Energy under contract EY-76-C-02-3077. Portions of the SETL code were borrowed from a program by Richard Kenner.

Code segment	Sequential	Hand compacted	Trace scheduled	Locally compacte
Floating add				
initialization	15.5	7.5	9.5	13
coefficient shift	3+3n	3+n	3+2n	3+2n
(n = no. of shift operations)				
add	9	6	6	8
Floating Multiply initialization	33	14	14	22
Floating divide				
initialization	19.5	8.6	7.5	14.5
loop	4	3	3.4	4
(for each of 48 iterations)				
normalize	10.5	4.5	5.5	7.5

Table 1. Weighted average execution time (in cycles) for sequential, hand compacted, trace scheduled, and locally compacted codes.

REFERENCES

[1] J.Fisher, "Trace Scheduling: A Technique for Global Microcode Compaction," IEEE Trans. Comput., vol. C-30, pp. 478-490, July 1981.

[2] D.Landskov, S.Davidson, B.Shriver, and B.Mallett, "Local Microcode Compaction Techniques," ACM Comput. Surveys, vol. 12, pp. 261-294, Sept. 1980.

[3] J.Fisher, D.Landskov, and B.Shriver, "Microcode Compaction: Looking Backward and Looking Forward," In Proc.1981 AFIPS National Computer Conf., AFIPS Press. Arlington. Va., pp. 95-102.

[4] R.B.K.Dewer, E.Schonberg and J.T.Schwartz, Higher level Programming. introduction to the Use of the Set-Theoretic Programming Language SETL, Courant Institute of Mathematical Sciences, Computer Science Department, New York University, 1981.

[5] R.Grishman, "The PUMA Project: Computer Design Automation in the University," Proc. 1980 Annl. Conf. ACM, pp. 490-497.

APPENDIX A

Shown below are three PUMA microcode segments used in our experiments, in their seqential, hand compacted, machine compacted (trace scheduled), and locally compacted forms. To limit our experiments, we have excluded some of the error-handling routines and other microcode sequences reterenced by these segments. In such cases we have indicated, as part of the sequential code, the registers which are "live" on entry to these routines.

```
*           PART 1: PRESHIFT
30          E0:BUF=XK
            AC=BUF
            E1:BUF=XJ
            MQ=BUF
            IF ILL(E0) THEN 30ILLEXP
            IF ILL(E1) THEN 30ILLEXP
            =E0-E1
            E2=E0-E1;IF EALU(11) THEN 30XKSMAL
30A         Y0=AC
            BUF=Y0
            AC=MQ
            IF AC(59) THEN 30NEGAC ELSE 30POSAC
30POSAC     MQ=0;GO TINYTEST
30NEGAC     MQ=-0;GO TINYTEST
30XKSMAL    AC=MQ;MQ=AC
            E2=7777-E2[F]
            E0=E1;GO 30A
TINYTEST    IF E2(6-11)=0 THEN 30TRY16
            IF ^E2(7-11)=0 THEN 30ADDZRO
            AC=MQ;MQ=AC
            AC:MQ=SHIFT(AC:MQ,A4);GO 30TRY16
30ADDZRO    AC=MQ;GO 30SAVEP6
*           PART 2: SHIFT
30TRY16     =E2;IF ^EALU(4)|EALU(5) THEN 30TRY4
            AC:MQ=SHIFT(AC:MQ,A16)
            E2=E2-20[F];GO 30TRY16
30TRY4      =E2;IF ^EALU(2)|EALU(3) THEN 30TRY1
            AC:MQ=SHIFT(AC:MQ,A4)
            E2=E2-4[F];GO 30TRY4
30TRY1      =E2;IF ^EALU(0)|EALU(1) THEN 30SHFTDN
            AC:MQ=SHIFT(AC:MQ,A1)
            E2=E2-1[F];GO 30TRY1
*           PART 3: POSTSHIFT
30SHFTDN    AC=MQ;MQ=AC
30SAVEP6    IF BUF(59) THEN 30NEGBF ELSE 30POSBF
30NEGBF     (AC)=AC-0[SAVEPG];GO 30ADD
30POSBF     (AC)=AC+0[SAVEPG] .
30ADD       AC=MQ;MQ=AC
            =AC+BUF[USEPG]
            AC=AC+BUF[USEPG];IF ^ALU(59)/ALU(48) THEN
+             WXIFLOAT
            =E0+1
            E0=E0+1
            AC=SHIFT(AC:MQ,A1);GO WXIFLOAT
WXIFLOAT    XI=E0:AC

30ILLEXP    * LIVE VARIABLES: AC, BUF, E0, E1
```

FLOATING ADD -- SEQUENTIAL CODE

```
30          E0:BUF=XK
30A         AC=BUF;E1:BUF=XJ;IF ILL(E0) THEN 30ILLEXP
30B         MQ=BUF;Y0=AC;IF ILL(E1) THEN 30ILLEXP
            AC=MQ;MQ=AC;=E0-E1
            E2=E0-E1;IF EALU(11) THEN 30XKSMAL
            BUF=Y0;MQ=0;IF AC(59) THEN 30NEGAC
TINYTEST    IF ^E2(6-11)=0 THEN 30ADDSML
30SHIFT     =E2;IF ^EALU(4)|EALU(5) THEN 30TRY4
30SH16      AC:MQ=SHIFT(AC:MQ,A16);E2=E2-20[F];
+           IF EALU(4)|EALU(5) THEN 30SH16
30TRY4      =E2;IF ^EALU(2)|EALU(3) THEN 30TRY1
30SH4       AC:MQ=SHIFT(AC:MQ,A4);E2=E2-4[F];
+           IF EALU(2)|EALU(3) THEN 30SH4
30TRY1      =E2;IF ^EALU(0)|EALU(1) THEN 30SHFTDN
30SH1       AC:MQ=SHIFT(AC:MQ,A1);E2=E2-1[F];
+           IF EALU(0)|EALU(1) THEN 30SH1
30SHFTDN    AC=MQ;MQ=AC;IF BUF(59) THEN 30NEGBUF
+             ELSE 30POSBUF
30NEGBUF    (AC)=AC-0[SAVEPG];GO 30ADD
30POSBUF    (AC)=AC+0[SAVEPG]
30ADD       AC=MQ;MQ=AC;IF OPCODE(2) THEN 30DP2
            =AC+BUF[USEPG];IF OPCODE(1) THEN 30DP
            NEWPARCEL;AC=AC+BUF[USEPG];=E0+1;
+           IF ^ALU(59)/ALU(48) THEN WXIFLOAT
            AC=SHIFT(AC:MQ,A1);E0=E0+1;GO WXIFLOAT
30ADDSML    MQ=AC;AC=MQ;IF ^E2(7-11)=0 THEN 30ADDZRO
            MQ=SHIFT(AC:MA,A4);=E2;IF EALU(4)|EALU(5)
+             THEN 30SH16 ELSE 30TRY4
30ADDZRO    MQ=AC;IF BUF(59) THEN 30NEGBUF ELSE 30POSBUF
30NEGAC     MQ=-0;IF E2(6-11) THEN 30SHIFT ELSE 30ADDSML
30XKSMAL    Y0=AC;AC=MQ;MQ=AC;E2=7777-E2[F]
            E0=E1;BUF=Y0;MQ=0;IF AC(59) THEN 30NEGAC
+             ELSE TINYTEST
```

FLOATING ADD -- HAND COMPACTED

```
*
*           PART 1: PRESHIFT
*
30          E0:BUF=XK
            AC=BUF;E1:BUF=XJ;IF ILL(E0) THEN 30ILLEXP
            MQ=BUF;=E0-E1
            IF ILL(E1) THEN 30ILLEXP
30A         E2=E0-E1;Y0=AC;IF EALU(11) THEN 30XKSMAL
NLAB10      BUF=Y0;AC=MQ
30POSAC     MQ=0;IF AC(59) THEN 30NEGAC ELSE TINYTEST
30NEGAC     MQ=-0;GO TINYTEST
30XKSMAL    AC=MQ;MQ=AC;E0=E1
            Y0=AC;E2=7777-E2[F];GO NLAB10
TINYTEST    IF E2(6-11) THEN 30TRY16
            IF ^E2(7-11) THEN 30ADDZRO
            AC=MQ;MQ=AC
            AC:MQ=SHIFT(AC:MQ,A4);GO 30TRY16
30ADDZRO    AC=MQ;GO 30SAVEP6
*
*           PART 2: SHIFT
*
30TRY16     =E2;IF ^EALU(4)|EALU(5) THEN 30TRY4
            E2=E2-20[F];AC:MQ=SHIFT(AC:MQ,A16);GO 30TRY16
30TRY4      =E2;IF ^EALU(2)|EALU(3) THEN 30TRY1
            E2=E2-4[F];AC:MQ=SHIFT(AC:MQ,A4);GO 30TRY4
30TRY1      =E2;IF ^EALU(0)|EALU(1) THEN 30SHFTDN
            E2=E2-1[F];AC:MQ=SHIFT(AC:MQ,A1);GO 30TRY1
*
*           PART 3: POSTSHIFT
*
30SHFTDN    AC=MQ;MQ=AC; =E0+1; IF BUF(59) THEN 30NEGBF ELSE
30POSBF
30SAVEP6    =E0+1;IF BUF(59) THEN 30NEGBF ELSE 30POSBF
30NEGBF     E0=E0+1;(AC)=AC-0[SAVEPG];GO 30ADD
30POSBF     E0=E0+1;(AC)=AC+0[SAVEPG]
30ADD       AC=MQ;MQ=AC
            =AC+BUF[USEPG]
            AC=AC+BUF[USEPG];IF ^ALU(59)/ALU(48) THEN
+             WXIFLOAT
            AC=SHIFT(AC:MQ,A1);GO WXIFLOAT
```

FLOATING ADD -- MACHINE COMPACTED

```
*               PART 1: PRESHIFT
30              E0:BUF=XK
                AC=BUF;E1:BUF=XJ
                MQ=BUF;IF ILL(E0) THEN 30ILLEXP
                IF ILL(E1) THEN 30ILLEXP
                =E0-E1
                E2=E0-E1;IF EALU(11) THEN 30XKSMAL
30A             Y0=AC
                BUF=Y0;AC=MQ
                IF AC(59) THEN 30NEGAC ELSE 30POSAC
30POSAC         MQ=0;GO TINYTEST
30NEGAC         MQ=-0;GO TINYTEST
30XKSMAL        AC=MQ;MQ=AC;E2=7777-E2[F]
                E0=E1;GO 30A
TINYTEST        IF E2(6-11)=0 THEN 30TRY16
                IF ^E2(7-11)=0 THEN 30ADDZRO
                AC=MQ;MQ=AC
                AC:MQ=SHIFT(AC:MQ,A4);GO 30TRY16
30ADDZRO        AC=MQ;GO 30SAVEP6
*               PART 2: SHIFT
30TRY16         =E2;IF ^EALU(4)|EALU(5) THEN 30TRY4
                AC:MQ=SHIFT(AC:MQ,A16);E2=E2-20[F];GO 30TRY16
30TRY4          =E2;IF ^EALU(2)|EALU(3) THEN 30TRY1
                AC:MQ=SHIFT(AC:MQ,A4);E2=E2-4[F];GO 30TRY4
30TRY1          =E2;IF ^EALU(0)|EALU(1) THEN 30SHFTDN
                AC:MQ=SHIFT(AC:MQ,A1);E2=E2-1[F];GO 30TRY1
*               PART 3: POSTSHIFT
30SHFTDN        AC=MQ;MQ=AC
30SAVEP6        IF BUF(59) THEN 30NEGBF ELSE 30POSBF
30NEGBF         (AC)=AC-0[SAVEPG];GO 30ADD
30POSBF         (AC)=AC-0[SAVEPG]
30ADD           AC=MQ;MQ=AC
                =AC+BUF[USEPG]
                AC=AC+BUF[USEPG];
+                 IF ^ALU(59)/ALU(48) THEN WXIFLOAT
                =E0+1
                E0=E0+1;AC=SHIFT(AC:MQ,A1);GO WXIFLOAT
```

FLOATING ADD -- LOCALLY COMPACTED

```
40          E1:BUF=XJ; IF REG(59) THEN 40XJNEG
            AC=BUF;GO 40GETXK
40XJNEG     AC=-BUF
40GETXK     E2:BUF=XK; IF REG(59) THEN 40XKNEG
            MQ=BUF;GO 40TESTILL
40XKNEG     MQ=-BUF
40TESTILL   IF ILL(E1) THEN 40ILLEXP
   .        IF ILL(E2) THEN 40ILLEXP
            Y1=AC
            AC=SHIFT(AC:MQ,L1)
            Y2=AC
            AC=SHIFT(AC:MQ,L1)
            Y4=AC
            AC=SHIFT(AC:MQ,L1)
            Y0=AC
            BUF=Y1
            =AC-BUF
            AC=AC-BUF
            Y7=AC
            BUF=Y2
            =AC-BUF
            AC=AC-BUF
            Y5=AC
            =AC-BUF
            AC=AC-BUF
            Y3=AC
            AC=SHIFT(AC:MQ,L1)
            Y6=AC
            IF ZERO(E1) THEN 40XJZERO
            IF ZERO(E2) THEN WXIZERO
            =E1+E2
            E0=E1+E2;IF XFOFL THEN FLRESFLO
40INTMUL    AC=0
            E2=15;GO EXIT
40XJZERO    IF ^ZERO(E2) THEN WXIZERON
            E0=6000;GO 40INTMUL

WXIZERO     * LIVE VARIABLES: NONE
FLRESFLO    * LIVE VARIABLES: E0
EXIT        * LIVE VARIABLES: Y0-Y7, MQ, AC, E0, E2
40ILLEXP    * LIVE VARIABLES: XJ, XK, E1, E2
```

INITIALIZATION OF FLOATING MULTIPLY
-- SEQUENTIAL CODE

```
40              E1:BUF=XJ;MQ=0;IF REG(59) THEN 40XJNEG
+                 ELSE 40XJPOS
40XJPOS         AC=BUF;E2:BUF=XK;IF ILL(E1) THEN 40ILLEXP
+               ELSE 44FORMMP
40XJNEG         AC=-BUF;E2:BUF=XK;IF ILL(E1) THEN 40ILLEXP
40FORMMP        Y1=AC;AC=SHIFT(AC:MQ,L1);IF ILL(E2) THEN
+                 40ILLEXP
                Y2=AC;AC=SHIFT(AC:MQ,L1);IF BUF(59) THEN 40XKNEG
                BUF=Y1;MQ=BUF;GO 40B
40XKNEG         BUF=Y1;MQ=-BUF
40B             Y4=AC;AC=SHIFT(AC:MQ,L1)
                Y0=AC;=AC-BUF
                BUF=Y2;AC=AC-BUF
                Y7=AC;=AC-BUF
                AC-BUF;IF ZERO(E1) THEN 40XJZERO
                Y5=AC;=AC-BUF;IF ZERO(E2) THEN WXIZERON
                AC=AC-BUF;=E1+E2
                Y3=AC;AC=SHIFT(AC:MQ,L1);E0=E1+E2;IF XFOFL
+               THEN FLRSFLON
40IMTMUL        Y6=AC;AC=0;E2=15
40XJZERO        Y5=AC;=AC-BUF;IF ^ZERO(E2) THEN WXIZERON
                AC=AC-BUF;IF ^OPCODE(1) THEN WXIZERON
                Y3=AC;AC=SHIFT(AC:MQ,L1);E0=6000;GO 40INTMUL
```

INITIALIZATION OF FLOATING MULTIPLY
-- HAND COMPACTED

```
40          E1:BUF=XJ;IF REG(59) THEN 40XJNEG
40GETXK     AC=BUF;E2:BUF=XK;IF REG(59) THEN 40XKNEG
40TESTILL   Y1=AC;MQ=BUF;IF ILL(E1) THEN 40ILLEXP
NLAB10      BUF=Y1;AC=SHIFT(AC:MQ,L1);IF ZERO(E2) THEN
+             WXIZERO
            Y2=AC;AC=SHIFT(AC:MQ,L1);IF ILL(E2) THEN
+             40ILLEXP
            Y4=AC;AC=SHIFT(AC:MQ,L1);IF ZERO(E1) THEN
+             40XJZERO
            Y0=AC;=AC-BUF;=E1+E2
            BUF=Y2;AC=AC-BUF;E0=E1+E2;IF XFOFL THEN FLRESFLO
            Y7=AC;=AC-BUF;E2=15
            AC=AC-BUF
            Y5=AC;=AC-BUF
            AC=AC-BUF
            Y3=AC;AC=SHIFT(AC:MQ,L1)
40INTMUL    Y6=AC;AC=0;GO EXIT
40XJZERO    Y0=AC;=AC-BUF;IF ^ZERO(E2) THEN WXIZERON
            BUF=Y2;AC=AC-BUF;E0=6000
            Y7=AC;=AC-BUF
            AC=AC-BUF
            Y5=AC;=AC-BUF
            AC=AC-BUF
            Y3=AC;AC=SHIFT(AC:MQ,L1)
40INTMUL_N  Y6=AC;AC=0;E2=15;GO EXIT
40XKNEG     Y1=AC;MQ=-BUF;IF ILL(E1) THEN 40ILLEXP ELSE
+             40XKNEG
40XJNEG     AC=-BUF;E2:BUF=XK;IF REG(59) THEN 40XKNEG
+           ELSE 40TESTILL
```

INITIALZATION OF FLOATING MULTIPLY
-- MACHINE COMPACTED

```
40          E1:BUF=XJ;IF REG(59) THEN 40XJNEG
            AC=BUF;GO 40GETXK
40XJNEG     AC=-BUF
40GETXK     E2:BUF=XK;IF REG(59) THEN 40XKNEG
            MQ=BUF;GO 40TESTILL
40XKNEG     MQ=-BUF
40TESTILL   IF ILL(E1) THEN 40ILLEXP
            IF ILL(E2) THEN 40ILLEXP
            Y1=AC;AC=SHIFT(AC:MQ,L1)
            Y2=AC;AC=SHIFT(AC:MQ,L1)
            Y4=AC;AC=SHIFT(AC:MQ,L1)
            BUF=Y1
            =AC-BUF;Y0=AC
            AC=AC-BUF;BUF=Y2
            =AC-BUF;Y7=AC
            AC=AC-BUF
            =AC-BUF;Y5=AC
            AC=AC-BUF
            AC=SHIFT(AC:MQ,L1);Y3=AC
            Y6=AC;IF ZERO(E1) THEN 40XJZERO
            IF ZERO(E2) THEN WXIZERO
            =E1+E2
            E0=E1+E2;IF XFOFL THEN FLRESFLO
40INTMUL    AC=0;E2=15;GO EXIT
40XJZERO    IF^ZERO(E2) THEN WXIZERON
            E0=6000;GO 40INTMUL
```

INITIALIZATION OF FLOATING MUTIPLY
-- LOCALLY COMPACTED

```
44          E2:BUF=XK; IF REG(59) THEN 44XKNEG
44XKPOS     AC=BUF
            Y0=AC
            E1:BUF=XJ
            AC=BUF;GO 44A
44XKNEG     AC=-BUF
            Y0=AC
            E1:BUF=XJ
            AC=-BUF
44A         Y1=AC
            IF ILL(E1) THEN 44ILLEXP
            IF ILL(E2) THEN 44ILLEXP
            IF ZERO(E1) THEN 44XJZERO
            IF ZERO(E2) THEN WXIINFN
            =E1-E2
            E0=E1-E2; IF XFOFL THEN FLRSFLON
            =E0-60
            E0=E0-60
            IF ^AC(59) THEN SKIP
            AC=-AC
SKIP        MQ=0
            AC:MQ=SHIFT(AC:MQ,A1)
            AC:MQ=SHIFT(AC:MQ,O1)
            BUF=Y0
44L         IF AC<<BUF&^MQ(49) THEN 44LOOP ELSE 44SUBTR
44LOOP      AC:MQ=SHIFT(AC:MQ,Z1);GO 44L
44SUBTR     =AC-BUF
            AC=AC-BUF; IF ALU(59) THEN 44READD
            IF MQ(50) THEN 44DONE
            AC:MQ=SHIFT(AC:MQ,O1);GO 44L
44DONE      AC=MQ
            IF MQ(49) THEN 44SHIFT
44X         IF OPCODE(0) THEN 44ROUND
44NORND     AC=SHIFT(AC:MQ,A1)
            NEWPARCEL
            BUF=Y1
            IF ^BUF(59) THEN 44POS
            AC=-AC
44POS       IF FOFL(E0) THEN FLRESFLO ELSE WXIFLOAT
WXIFLOAT    XI=E0:AC;GO NEWINSTR
44SHIFT     =E0+1
            E0=E0+1
            AC=SHIFT(AC:MQ,A1);GO 44X
44ROUND     (AC)=AC+0[NOP]
            AC=AC+0[NOP];GO 44NORND
44XJZERO    NEWPARCEL
            IF ZERO(E2) THEN WXIINDEF ELSE WXIZERO
44READD     =AC+BUF
            AC=AC+BUF;GO 44LOOP
```

FLOATING DIVISION -- SEQUENTIAL CODE

```
NEWINSTR     * LIVE VARIABLES: NONE
WXIINDEF     * LIVE VARIABLES: NONE
WXIINFN      * LIVE VARIABLES: MQ, AC
WXIZERO      * LIVE VARIABLES: NONE
FLRSFLON     * LIVE VARIABLES: E0
FLRESFLO     * LIVE VARIABLES: E0
44ILLEXP     * LIVE VARIABLES: XJ, XK, E1, E2
```

FLOATING DIVISION -- SEQUENTIAL CODE
(CONTINUED)

```
44          E2:BUF=XK;IF REG(59) THEN 44XKNEG
44XKPOS     AC=BUF;E1:BUF=XJ;IF ILL(E2) THEN 44ILLEXP
            Y0=AC;AC=BUF;IF ILL(E1) THEN 44ILLEXP ELSE 44A
44XKNEG     AC=-BUF;E1:BUF=XJ;IF ILL(E2) THEN 44ILLEXP
            Y0=AC;AC=-BUF;IF ILL(E1) THEN 44ILLEXP
44A         Y1=AC;MQ=0;IF ZERO(E1) THEN 44XJZERO
            BUF=Y0;IF ZERO(E2) THEN WXIINFN
            =E1-E2;IF AC(59) THEN 44COMP
            E0=E1-E2;IF XFOFL THEN FLRFLON ELSE 44B
44COMP      E0=E1-E2;AC=-AC;IF XFOFL THEN FLRSFLON
44B         =E0-60;AC:MQ=SHIFT(AC:MQ,A1)
            E0=E0-60;AC:MQ=SHIFT(AC:MQ,O1);
+           IF AC<<BUF&^MQ(49) THEN 44LOOP ELSE 44SUBTR
44LOOP      AC:MQ=SHIFT(AC:MQ,Z1);IF AC<<BUF&^MQ(49) THEN
+             44LOOP
44SUBTR     =AC-BUF;IF MQ(50) THEN 44DONE
            AC=AC-BUF;IF ALU(59) THEN 44READD
            AC:MQ=SHIFT(AC:MQ,O1);IF AC<<BUF&^MQ(49)
+           THEN 44LOOP ELSE 44SUBTR
44READD     =AC+BUF
            AC=AC+BUF;GO 44LOOP
44DONE      AC=MQ;BUF=Y1;IF MQ(49) THEN 44SHIFT
            (AC)=AC+0[NOP];IF OPCODE(0) THEN 44ROUND
44NORND     NEWPARCEL;AC=SHIFT(AC:MQ,A1);IF BUF(59) THEN
+             44NEGRES
            XI=E0:AC;IF FOFL(E0) THEN FLRESFLO ELSE NEWINSTR
44NEGRES    AC=-AC;IF FOFL(E0) THEN FLRESFLO ELSE WXIFLOAT
44SHIFT     =E0+1;AC=SHIFT(AC:MQ,A1)
            (AC)=AC+0[NOP];E0=E0+1;IF OPCODE(0) THEN 44ROUND
+           ELSE 44NORND
44ROUND     AC=AC+0[NOP];GO 44NORND
44XJZERO    NEWPARCEL;IF ZERO(E2) THEN WXIINDEF ELSE WXIZERO
```

FLOATING DIVISION -- HAND COMPACTED

```
44          E2:BUF=XK;IF REG(59) THEN 44XKNEG
44XKPOS     AC=BUF;E1:BUF=XJ;IF ZERO(E2) THEN NLAB5
            Y0=AC;AC=BUF;IF ILL(E2) THEN 44ILLEXP ELSE
+             NLAB15
44XKNEG     AC=BUF;E1:BUF=XJ;IF ILL(E2) THEN 44ILLEXP
            Y0=AC;AC=-BUF;IF ZERO(E2) THEN WXIINFN
NLAB15      =E1-E2;BUF=Y0;IF ^AC(59) THEN SKIP
            AC=-AC;Y1=AC
            MQ=0;GO NLAB22
SKIP        Y1=AC;MQ=0;IF ILL(E1) THEN 44ILLEXP
NLAB22      AC:MQ=SHIFT(AC:MQ,A1);IF ZERO(E1) THEN 44XJZERO
            AC:MQ=SHIFT(AC:MQ,O1);E0=E1-E2;IF XFOFL THEN
+             FLRSFLON
44L         =AC-BUF;IF AC<<BUF&^MQ(49) THEN 44LOOP
            AC=AC-BUF;IF ALU(59) THEN 44READD
            IF MQ(50) THEN 44DONE
            AC:MQ=SHIFT(AC:MQ,O1);GO 44L
44LOOP      AC:MQ=SHIFT(AC:MQ,Z1);GO 44L
44DONE      AC=MQ;BUF=Y1;=E0-60;IF MQ(49) THEN NLAB43
44X         E0=E0-60;NEWPARCEL;IF OPCODE(0) THEN 44ROUND_N
44NORND     AC=SHIFT(AC:MQ,A1);IF ^BUF(59) THEN 44POS
            AC=-AC
44POS       XI=E0:AC;IF FOFL(E0) THEN FLRESFLO ELSE NEWINSTR
NLAB43      AC=SHIFT(AC:MQ,A1);E0=E0-60
44SHIFT     =E0+1;NEWPARCEL;IF OPCODE(0) THEN 44ROUND
            E0=E0+1;GO 44NORND
44ROUND     (AC)=AC+0[NOP];E0=E0+1
NLAB45      AC=AC+0[NOP];GO 44NORND
44XJZERO    NEWPARCEL;IF ZERO(E2) THEN WXIINDEF ELSE WXIZERO
44READD     =AC+BUF
            AC=AC+BUF;GO 40LOOP
NLAB5       AC=BUF;GO WXIINEF
44ROUND_N   (AC)=AC+0[NOP];GO NLAB45
```

FLOATING DIVISION -- MACHINE COMPACTED

```
44          E2:BUF=XK;IF REG(59) THEN 44XKNEG
44XKPOS     AC=BUF;E1:BUF=XJ
            Y0=AC;AC=BUF;GO 44A
44XKNEG     AC=-BUF;E1:BUF=XJ
            Y0=AC;AC=-BUF
44A         Y1=AC;IF ILL(E1) THEN 44ILLEXP
            IF ILL(E2) THEN 44ILLEXP
            IF ZERO(E1) THEN 44XJZERO
            IF ZERO(E2) THEN WXIINFN
            =E1-E2
            E0=E1-E2;IF XFOFL THEN FLRSFLON
            =E0-60
            E0=E0-60;IF ^AC(59) THEN SKIP
            AC=-AC
SKIP        MQ=0
            AC:MQ=SHIFT(AC:MQ,A1)
            AC:MQ=SHIFT(AC:MQ,O1);BUF=Y0
44L         IF AC<<BUF&^MQ(49) THEN 44LOOP ELSE 44SUBTR
44LOOP      AC:MQ=SHIFT(AC:MQ,Z1);GO 44L
44SUBTR     =AC-BUF
            AC=AC-BUF;IF ALU(59) THEN 44READD
            IF MQ(50) THEN 44DONE
            AC:MQ=SHIFT(AC:MQ,Z1);GO 44L
44DONE      AC=MQ;IF MQ(49) THEN 44SHIFT
44X         IF OPCODE(0) THEN 44ROUND
44NORND     AC=SHIFT(AC:MQ,A1);NEWPARCEL;BUF=Y1
            IF ^BUF(59) THEN 44POS
            AC=-AC
44POS       IF FOFL(E0) THEN FLRESFLO ELSE WXIFLOAT
WXIFLOAT    XI=E0:AC;GO NEWINSTR
44SHIFT     =E0+1
            E0=E0+1;AC=SHIFT(AC:MQ,A1);GO 44X
44ROUND     (AC)=AC+0[NOP]
            AC=AC+0[NOP]; GO 44NORND
44XJZERO    NEWPARCEL;IF ZERO(E2) THEN WXIINDEF ELSE WXIZERO
44READD     =AC+BUF
            AC=AC+BUF;GO 44LOOP
```

FLOATING DIVISION -- LOCALLY COMPACTED

APPENDIX B

Given below is a detailed comparison of the timings of the codes shown in Appendix A. Each row gives the time for one possible path through the code. The first two or three columns specify the values of conditions for which that path is taken; the last four columns give the timings for the four versions of the code. The last page of this appendix lists the space requirements of the various codes.

FLOATING ADD

PART 1: PRESHIFT

(A) MAIN PATH:

NEG.	XK SMALL	EXP SMALL	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
0	0	0	14	7	8	12
0	0	1	17	9*	11	14
0	1	0	17	8	10	14
0	1	1	20	11*	13	16
1	0	0	14	7	9	12
1	0	1	17	9*	12	14
1	1	0	17	8	11	14
1	1	1	20	11*	14	16

* shift part is 1 cycle shorter for these paths

(B) ADDZERO PATH (one operand = 0):

NEG.	XK SMALL	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
0	0	15	8	9	14
0	1	18	9	11	16
1	0	15	8	10	14
1	1	18	9	13	16

PART 2: SHIFT

	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
INSIDE LOOP	3*3	3*1	3*2	3*2
OUTSIDE LOOP	3*0	3*1	3*0	3*0

PART 3: POST SHIFT

OVERFLOW	SUM NEG.	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
0	0	9	6	6	8
0	1	9	6	6	8
1	0	6	5	5	6
1	1	6	5	5	6

INITIALIZATION OF FLOATING MULTIPLY

XJ ZERO	XJ NEG.	XK NEG.	SEQUENTIAL CODE	HAND COMPACTED	FISHER'S COMPACTED	MACHINE* COMPACTED	LOCALLY COMPACTED
0	0	0	33	14	14	14	22
0	0	1	33	14	14	14	22
0	1	0	33	14	14	14	22
0	1	1	33	14	14	14	22
1	0	0	32	14	16	14	22
1	0	1	32	14	16	14	22
1	1	0	32	14	16	14	22
1	1	1	32	14	16	14	22

* Because our implementation did not include space saving, our version was somewhat longer than Fisher's but was faster for the (relatively rare) case XJ=0.

FLOATING DIVISION

PART 1: PRE-LOOP

XK NEG.	XJ ZERO	XJ NEG.	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
0	0	0	19	9*	7	14
0	0	1	20	9*	8	15
0	1	-	11	4	7	7
1	0	0	19	9*	7	14
1	0	1	20	9*	8	15
1	1	-	11	4	7	7

* these paths will be 1 cycle shorter when AC<<BUF|^MQ(49)

PART 2: LOOP

	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
MAIN LOOP	5	4	4	5
READD	6	5	5	6
AC<<BUF|^MQ(49)	2	1	2	2

PART 3: POST-LOOP

QUOTIENT NEG.	SHIFT	ROUND	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
0	0	0	9	4	4	6
0	0	1	12	5	6	8
0	1	0	11	5	6	8
0	1	1	14	6	7	10
1	0	0	10	4	5	7
1	0	1	13	5	7	9
1	1	0	12	5	7	9
1	1	1	15	6	8	11

SPACE REQUIREMENTS

FLOATING ADD

	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
PRE SHIFT	22	15	15	18
SHIFT	10	6	6	6
POST SHIFT	10	7	8	9
TOTAL SPACE (LINES)	42	28	29	33

INITIALIZATION OF FLOATING MULTIPLY

	SEQUENTIAL CODE	HAND COMPACTED	FISHER'S COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
TOTAL SPACE (LINES)	36	19	19	22	26

FLOATING DIVISION

	SEQUENTIAL CODE	HAND COMPACTED	MACHINE COMPACTED	LOCALLY COMPACTED
PRE LOOP	26	12	13	17
LOOP	8	6	7	8
POST LOOP	15	9	11	11
TOTAL SPACE (LINES)	49	27	31	36

NYU COMPSCI TR-043
Grishman, Ralph c.1

A preliminary
evaluation of ...